

Parallel Particle Iterators

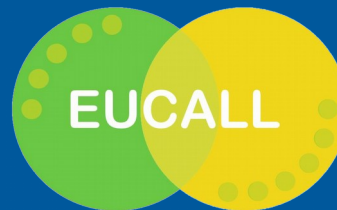
A.Huebl¹, R.Widera¹, H. Burau^{1,2}, S.T. Hahn^{1,2}

¹ Helmholtz-Zentrum Dresden - Rossendorf

² Technische Universität Dresden

HZDR, Dresden

11.05.17



PICongGPU Datenstrukturen

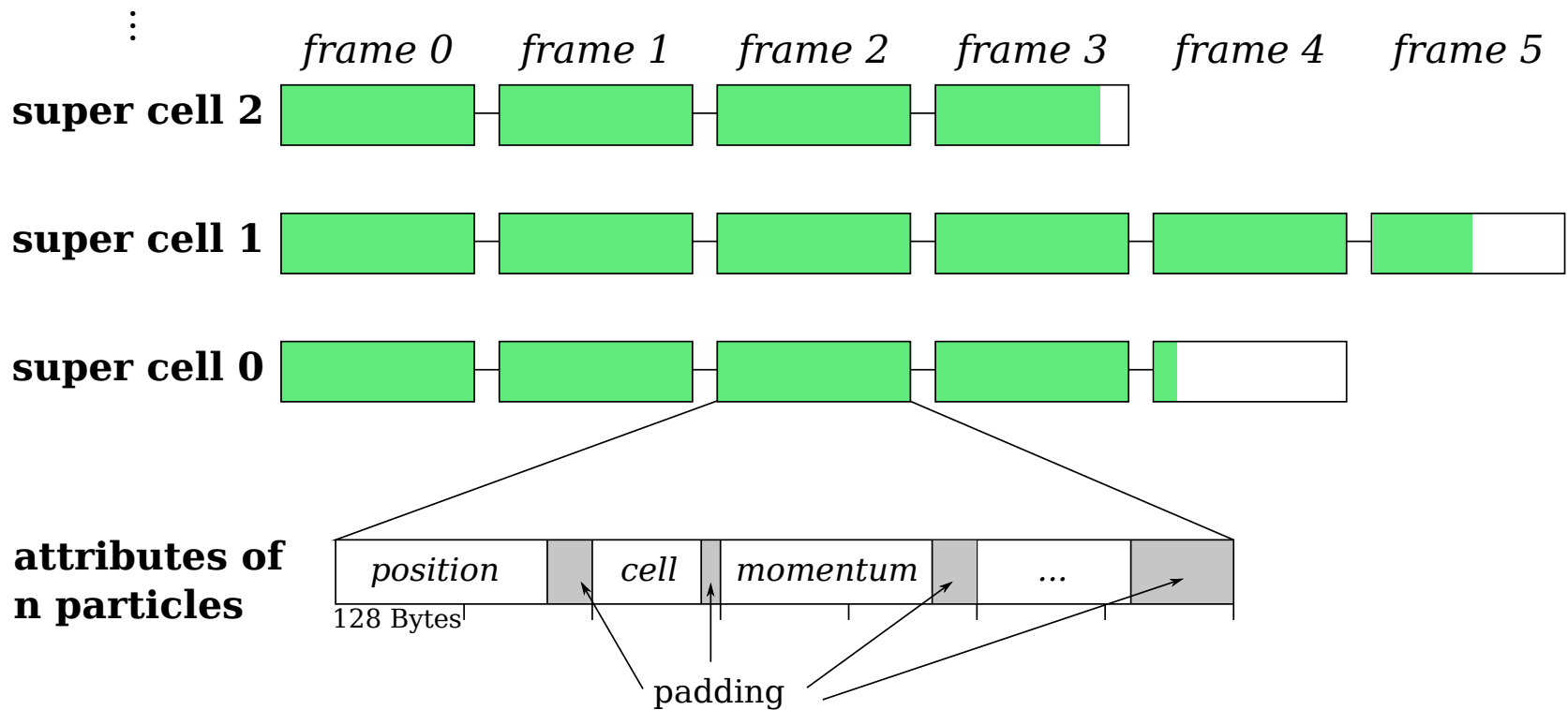


Fig: A. Huebl , Diploma Thesis,
DOI:10.5281/zenodo.15924 (2014)

Zielstellung

- Algorithmen sollen auf verschiedenen Datenstrukturen arbeiten
 - Einmaliges schreiben der Algorithmen,
 - Austausch der Datenstrukturen
- Bereitstellen eines Iterators für Datenzugriff
 - Einfache Benutzung
 - Gleichbleibendes Interface
 - Großer Funktionsumfang
 - ✓ Bestimmbarkeit der Iterationsrichtung: Vorwärts, Rückwärts
 - ✓ Parallele/ nicht Parallele Ausführung
 - ✓ Unterstützung hierarchischer heterogener Datenstrukturen



Views

Views

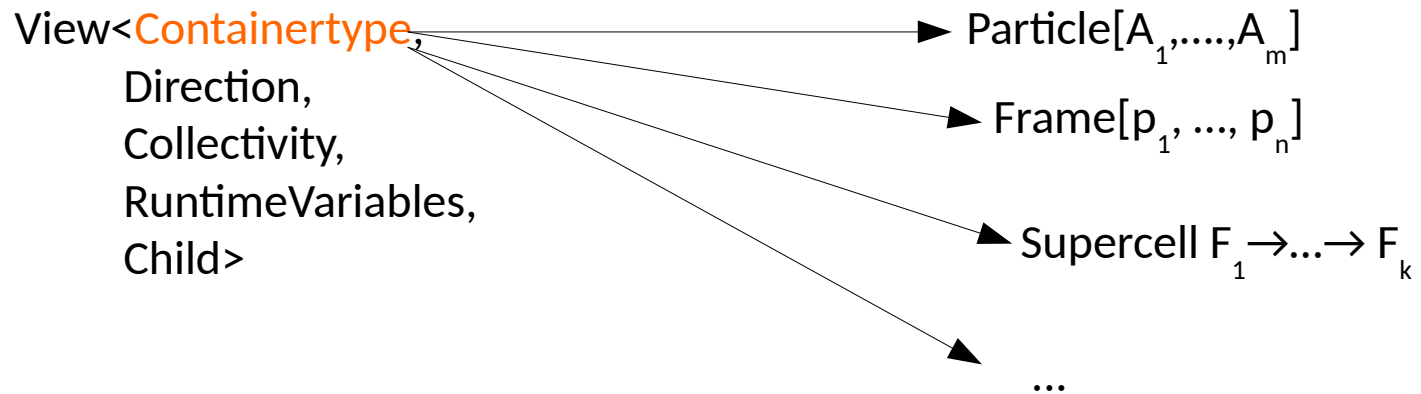
- Kapseln die Datencontainer
- Stellen `begin()` und `end()` bereit

`View<Containertype,
Direction,
Collectivity,
RuntimeVariables,
Child>`



Views: Datatype

- Kapseln die Datencontainer
- Stellen `begin()` und `end()` bereit



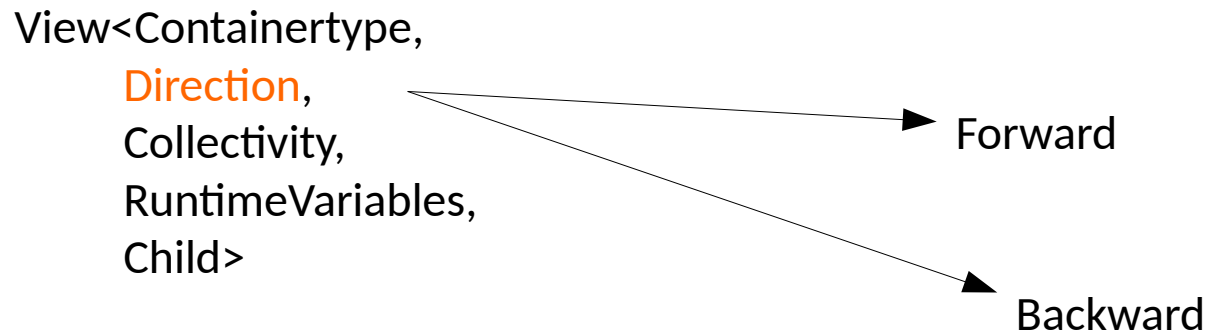
Anforderung:

- Trait Componenttyp hat `Containertype` Ausprägung



Views: Direction

- Kapseln die Datencontainer
- Stellen begin() und end() bereit



Views Collectivity

- Kapseln die Datencontainer
- Stellen `begin()` und `end()` bereit

View<Containertype,
Direction,
Collectivity,
RuntimeVariables,
Child>  None

Klasse mit 2 Funktionen:

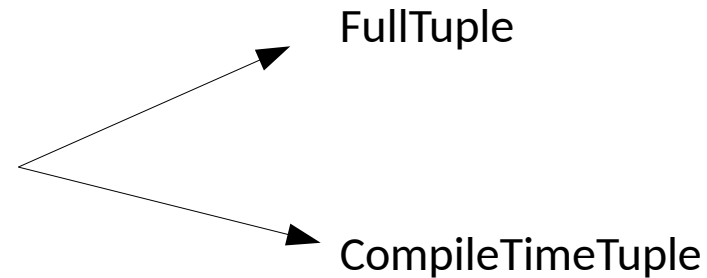
- `void sync()`
- `bool isMover()`



Views: Runtime

- Kapseln die Datencontainer
- Stellen begin() und end() bereit

View<Containertype,
Direction,
Collectivity,
RuntimeTuple,
Child>



Vorraussetzungen:

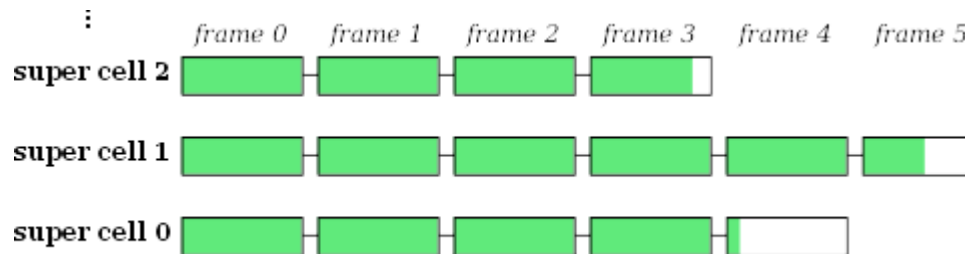
- 1.uint getOffset(): Erstes benutztes Element
- 2.uint getNbElements(): Anzahl Elemente in Datenstruktur
- 3.uint getJumpsize(): Abstand zum nächsten Element



RuntimeTuple.getNbElements()

- Zwei Optionen die Anzahl Elemente zu bestimmen:
 - 1) Templateparameter
 - 2) Laufzeitvariable
- Trait **NeedRuntimeSize** entscheidet welche Optionen gewählt wird:
 - 1) True: getNbElements() wird benötigt
 - 2) False: Trait **NumberElements** wird benutzt

Beispiel Particles in Frame



Views: Abbildung von Hierarchien

- Kapseln die Datencontainer
- Stellen `begin()` und `end()` bereit

View<Containertype,
Direction,
Collectivity,
RuntimeVariables,
Child>

NoChild

View<....>

Idee:

- Jede Hierarchieebene ist eine View
- Verschachteln der Datenstrukturen



Views: Der Rückgabetype

Problem:

✗ Ungültiges Element
✗ Nachfolger Element ist
gültig

Lösung:

✓ Abfrage ob Element gültig ist

Beispiel:

```
View<Containertype,  
    Direction,  
    Collectivity,  
    RuntimeVariables,  
    Child>view;
```

```
auto it=view.begin();  
auto wrapper =*it;  
if(wrapper)  
{  
    std::cout << *wrapper;  
}
```



Beispiele

Get Smart

Beispiel: Alle Particle in einem Frame

```
uint jumpsize=1;
uint offset=0;
uint nbElem=10;
RuntimeTuple runtimeFrame(offset, nbElem, jumpsize);

View<Frame, Direction::Forward,
    Collectivity::None, RuntimeTuple>
    view(frame, runtimeVar);
std::cout << "Frame before calculation" << frame << std::endl;

for(auto it = view.begin(); it!=view.end(); ++it){
    auto wrapper = *it;
    if(wrapper){
        (*wrapper).data[0] *=2;
        (*wrapper).data[1] *=3;
    }
}

std::cout << std::endl << "Frame after calculation" << frame;
```



Beispiel: Alle Particle in einem Frame

Frame before calculation:

$[(0, 1), (2, 3), (4, 5), (6, 7), (8, 9)]$

Frame after calculation:

$[(0, 3), (4, 9), (8, 15), (12, 21), (16, 27)]$



Beispiel: Jedes zweite Frame in einer Superzelle

```
View<Supercell, Direction::Backward, Collectivity::None, RuntimeTuple>  
    view(supercell, runtimeSupercell);
```

```
const auto jumpsize=2;  
const auto offset = 0;  
const auto nbElements = -1;
```

```
RuntimeTuple runtimeSupercell(offset, jumpsize, nbElements);  
std::cout << "Supercell complete:" << std::endl;  
std::cout << supercell << std::endl;  
std::cout << std::endl << "Every second Frame:" << std::endl;
```

```
for(auto it = view.begin(); it!=view.end(); ++it)  
{  
    if(*it)  
    {  
        std::cout << "Frame:" << **it << std::endl;  
    }  
}
```



Beispiel: Jedes zweite Frame in einer Superzelle

Supercell complete:

[(0, 1), (2, 3), (4, 5), (6, 7), (8, 9)]
[(20, 21), (22, 23), (24, 25), (26, 27), (28, 29)]
[(40, 41), (42, 43), (44, 45), (46, 47), (48, 49)]
[(60, 61), (62, 63), (64, 65), (66, 67), (68, 69)]
[(80, 81), (82, 83), (-1, -1), (-1, -1), (-1, -1)]

Every second frame:

Frame: [(0, 1), (2, 3), (4, 5), (6, 7), (8, 9)]
Frame: [(40, 41), (42, 43), (44, 45), (46, 47), (48, 49)]
Frame: [(80, 81), (82, 83), (-1, -1), (-1, -1), (-1, -1)]



Beispiel: Particle jedes zweiten Frames einer Supercelle

```
typedef View<Frame,  
    Direction::Forward,  
    Collectivity::None,  
    RuntimeTuple> PartInFrame;  
  
View<Supercell, Direction::Backward, Collectivity::None, RuntimeTuple, PartInFrame>  
    view(supercell, RuntimeTuple(0, -1, 2) , PartInFrame(nullptr, RuntimeTuple(0,2,1)));  
  
Std::cout << "Supercell" << supercell << std::endl;  
Std::cout << "Particles: ";  
  
for(auto it = view.begin(); it!=view.end(); ++it)  
{  
    if(*it)  
    {  
        std::cout << **it << ", ";  
    }  
}
```



Beispiel: Particle jedes zweiten Frames einer Supercelle

Supercell:

[(0, 1), (2, 3), (4, 5), (6, 7), (8, 9)]

[(20, 21), (22, 23), (24, 25), (26, 27), (28, 29)]

[(40, 41), (42, 43), (44, 45), (46, 47), (48, 49)]

[(60, 61), (62, 63), (64, 65), (66, 67), (68, 69)]

[(80, 81), (82, 83), (-1, -1), (-1, -1), (-1, -1)]

Particles: (0, 1), (2, 3), (4, 5), (6, 7), (8, 9), (40, 41), (42, 43), (44, 45),
(46, 47), (48, 49), (80, 81), (82, 83)



Beispiel: Particle-Particle Interaktionen

```
hzdr::View<...> iterSuperCell1(supercellContainer[0], runtimeSupercell1,  
                               ParticleInFrameView(nullptr, runtimeVarParticle1));  
  
hzdr::View<...> iterSuperCell2(supercellContainer[1], runtimeSupercell2,  
                               ParticleInFrameView(nullptr, runtimeVarParticle2));  
  
std::cout << "First Supercell" << supercellContainer[0] << std::endl << std::endl;  
std::cout << "Second Supercell" << supercellContainer[1] << std::endl;
```

```
for(auto it=iterSuperCell1.begin(); it != iterSuperCell1.end(); ++it)  
{  
    if(*it) (**it).data[0] = 0;  
    for(auto it2 = iterSuperCell2.begin(); it2 != iterSuperCell2.end(); ++it2)  
    {  
        // check wheter both are valid  
        if(*it and *it2)  
        {  
            (**it).data[0] += (**it2).data[1];  
        }  
    }  
}
```

```
Std::cout << "First Supercell after:" << supercellContainer[0] << std::endl << std::endl;
```



First Supercell

[(0, 1), (2, 3), (4, 5), (6, 7), (8, 9),]
[(10, 11), (12, 13), (14, 15), (16, 17), (18, 19),]
[(20, 21), (22, 23), (24, 25), (26, 27), (28, 29),]
[(30, 31), (32, 33), (34, 35), (36, 37), (38, 39),]
[(40, 41), (42, 43), (-1, -1), (-1, -1), (-1, -1),]

Second Supercell

[(50, 51), (52, 53), (54, 55), (56, 57), (58, 59),]
[(60, 61), (62, 63), (64, 65), (66, 67), (68, 69),]
[(70, 71), (72, 73), (74, 75), (76, 77), (78, 79),]
[(80, 81), (82, 83), (84, 85), (86, 87), (-1, -1),]



Beispiel: Particle-Particle Interaktionen

```
hzdr::View<...> iterSuperCell1(supercellContainer[0], runtimeSupercell1,  
    ParticleInFrameView(nullptr, runtimeVarParticle1));  
  
for(auto it=iterSuperCell1.begin(); it != iterSuperCell1.end(); ++it)  
{  
    for(auto it2 = iterSuperCell1.begin(); it2 != iterSuperCell1.end(); ++it2)  
    {  
        // check wheter both are valid  
        if(*it and *it2)  
        {  
            if((*it).data[0] != (*it2).data[0])  
                std::cerr << "Error";  
        }  
    }  
}
```



Beispiel: Particle-Particle Interaktionen

```
hzdr::View<...> iterSuperCell1(supercellContainer[0], runtimeSupercell1,  
    ParticleInFrameView(nullptr, runtimeVarParticle1));  
  
for(auto it=iterSuperCell1.begin(); it != iterSuperCell1.end(); ++it)  
{  
    for(auto it2 = iterSuperCell1.begin(); it2 != iterSuperCell1.end(); ++it2)  
    {  
        // check wheter both are valid  
        if(*it and *it2)  
        {  
            if((*it).data[0] != (*it2).data[0])  
                std::cerr << "Error";  
        }  
    }  
}
```




```

std::cout << "Supercell " << cell << std::endl;
const uint nbThreads = 5;
int count = 0;
for(int id=0; id<nbThreads; ++id)
{
    View<Supercell, hzdr::Direction::Forward, hzdr::Collectivity::None,
        RuntimeTuple, ParticleInFrame> iterSuperCell(cell, runtimeSupercell2,
            ParticleInFrame(nullptr, RuntimeTuple(id, cell.nbElemsLastFrame, nbThreads));
    for(auto it=iterSuperCell.begin(); it != iterSuperCell.end(); ++it)
    {
        if(*it)
        {
            (**it).data[0] = id;
        }
        else
        {
            ++count;
        }
    }
}

std::cout << "Number of invalids " << count << std::endl;
std::cout << "First Supercell after Calc" << cell << std::endl;

```



Noch offene Fragen?